TLA$^+$ Video Course – Lecture 10, Part 2

Leslie Lamport

IMPLEMENTATION WITH REFINEMENT

REFINEMENT MAPPINGS

This video should be viewed in conjunction with a Web page.
To find that page, search the Web for *TLA+ Video Course* .

The TLA$^+$ Video Course
Lecture 10
Implementation with Refinement

Having finished the preliminaries, we head to our main goal: understanding what it means in general for one specification to implement another, and how we can check that it does. We will take a rather long path, and it may not always be clear where it's leading. But just follow it step by step. The destination is worth the effort.

AB2 IMPLEMENTS AB

The $AB2$ protocol doesn't just implement the $ABSpec$ spec.

The $AB2$ protocol doesn't just implement the high level spec of module $ABSpec$.

The $AB2$ protocol doesn't just implement the $ABSpec$ spec.

It implements the AB protocol, where a $LoseMsg$ step of AB is implemented by a $CorruptMsg$ step of $AB2$.

The $AB2$ protocol doesn't just implement the high level spec of module $ABSpec$.

It actually implements the AB protocol, where an AB protocol step that loses a message is implemented by the $AB2$ protocol step that corrupts the message.

The $AB2$ protocol doesn't just implement the $ABSpec$ spec.

It implements the AB protocol, where a $LoseMsg$ step of AB is implemented by a $CorruptMsg$ step of $AB2$.

Programmers find this confusing.

The $AB2$ protocol doesn't just implement the high level spec of module $ABSpec$.

It actually implements the AB protocol, where an AB protocol step that loses a message is implemented by the $AB2$ protocol step that corrupts the message.

Most programmers will find this confusing.

The $AB2$ protocol doesn't just implement the $ABSpec$ spec.

It implements the AB protocol, where a $LoseMsg$ step of AB is implemented by a $CorruptMsg$ step of $AB2$.

Programmers find this confusing.

They don't think losing a message is a step of the AB protocol, but rather a step of the environment.

They don't think of losing a message as a step of the AB protocol, but rather as a step taken by the environment in which the protocol is executed.

Our specifications say nothing about who performs what steps.

Our specifications say nothing about who performs what steps.

Our specifications say nothing about who performs what steps.

We think that in the AB spec:

Sender A performs $ASnd$ and $ARcv$ steps.

Our specifications say nothing about who performs what steps.

We think that in the AB protocol spec: Sender A performs $ASnd$ and $ARcv$ steps.

Our specifications say nothing about who performs what steps.

We think that in the AB spec:

Sender A performs $ASnd$ and $ARcv$ steps.

Receiver B performs $BSnd$ and $BRcv$ steps.

Our specifications say nothing about who performs what steps.

We think that in the AB protocol spec: Sender A performs $ASnd$ and $ARcv$ steps.

Receiver B performs $BSnd$ and $BRcv$ steps.

Our specifications say nothing about who performs what steps.

We think that in the AB spec:

Sender A performs $ASnd$ and $ARcv$ steps.

Receiver B performs $BSnd$ and $BRcv$ steps.

The communication infrastructure performs $LoseMsg$ steps.

Our specifications say nothing about who performs what steps.

We think that in the AB protocol spec: Sender A performs $ASnd$ and $ARcv$ steps.
Receiver B performs $BSnd$ and $BRcv$ steps.
And the communication infrastructure performs $LoseMsg$ steps.

Our specifications say nothing about who performs what steps.

We think that in the AB spec:

Sender A performs $ASnd$ and $ARcv$ steps.

Receiver B performs $BSnd$ and $BRcv$ steps.

The communication infrastructure performs $LoseMsg$ steps.

That's just an interpretation we put on the spec.

But that's just an interpretation that we put on the spec, suggested by the way we write the next-state action as the disjunction of subactions.

Our specifications say nothing about who performs what steps.

We think that in the AB spec:

Sender A performs $ASnd$ and $ARcv$ steps.

Receiver B performs $BSnd$ and $BRcv$ steps.

The communication infrastructure performs $LoseMsg$ steps.

That's just an interpretation we put on the spec.

It would be easy to make the spec suggest a different interpretation.

But that's just an interpretation that we put on the spec, suggested by the way we write the next-state action as the disjunction of subactions.

It would be easy to make the spec suggest a different interpretation–for example by decomposing the next-state action to suggest that A and B both *send* messages and cause the messages to be lost.

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

The $AB2$ protocol implements the AB protocol,
where $CorruptMsg$ steps implement $LoseMsg$ steps.

The goal: convince ourselves that this is true.

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

Our goal goal now is to convince ourselves that this is true.

The $AB2$ protocol implements the AB protocol,
where $CorruptMsg$ steps implement $LoseMsg$ steps.

The goal: convince ourselves that this is true.

This requires answering two questions:

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

Our goal goal now is to convince ourselves that this is true.

Reaching it requires answering two questions:

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

The goal: convince ourselves that this is true.

This requires answering two questions:

1. What does it mean?

The first is: What does it mean?

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

The goal: convince ourselves that this is true.

This requires answering two questions:

1. What does it mean?
2. How do we check it?

The first is: What does it mean?

And the second is: How do we check it?

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

First, exactly what does this mean?

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

First, exactly what does this mean?

It means that for every behavior of the $AB2$ protocol we can obtain a behavior of the AB protocol by changing the state as shown in the following example:

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

State of $AB2$

$$\begin{aligned} AVar &= \langle \text{“}Tom\text{”}, 1 \rangle \\ BVar &= \langle \text{“}Ann\text{”}, 0 \rangle \\ AtoB2 &= \langle Bad, \langle \text{“}Tom\text{”}, 1 \rangle \rangle \\ BtoA2 &= \langle 0, Bad, 0, Bad \rangle \end{aligned}$$

For this state in a behavior of $AB2$

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

<u>State of $AB2$</u>		<u>State of AB</u>
$AVar = \langle \text{“}Tom\text{”}, 1 \rangle$		$AVar =$
$BVar = \langle \text{“}Ann\text{”}, 0 \rangle$	$\longrightarrow$	$BVar =$
$AtoB2 = \langle Bad, \langle \text{“}Tom\text{”}, 1 \rangle \rangle$		$AtoB =$
$BtoA2 = \langle 0, Bad, 0, Bad \rangle$		$BtoA =$

For this state in a behavior of $AB2$ here's how we get the corresponding state in a behavior of AB.

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

State of $AB2$		State of AB
$AVar = \langle \text{"}Tom\text{"}, 1 \rangle$		$AVar = \langle \text{"}Tom\text{"}, 1 \rangle$
$BVar = \langle \text{"}Ann\text{"}, 0 \rangle$	$\longrightarrow$	$BVar = \langle \text{"}Ann\text{"}, 0 \rangle$
$AtoB2 = \langle Bad, \langle \text{"}Tom\text{"}, 1 \rangle \rangle$		$AtoB =$
$BtoA2 = \langle 0, Bad, 0, Bad \rangle$		$BtoA =$

For this state in a behavior of $AB2$ here's how we get the corresponding state in a behavior of AB.

The values of $AVar$ and $BVar$ are the same.

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

State of $AB2$		State of AB
$AVar = \langle \text{"}Tom\text{"}, 1 \rangle$		$AVar = \langle \text{"}Tom\text{"}, 1 \rangle$
$BVar = \langle \text{"}Ann\text{"}, 0 \rangle$	$\longrightarrow$	$BVar = \langle \text{"}Ann\text{"}, 0 \rangle$
$AtoB2 = \langle Bad, \langle \text{"}Tom\text{"}, 1 \rangle \rangle$		$AtoB =$
$BtoA2 = \langle 0, Bad, 0, Bad \rangle$		$BtoA =$

For this state in a behavior of $AB2$ here's how we get the corresponding state in a behavior of AB.

The values of $AVar$ and $BVar$ are the same.

We obtain the sequence of messages $AtoB$ from the sequence of messages $AtoB2$

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

State of $AB2$		State of AB
$AVar = \langle \text{“}Tom\text{”}, 1 \rangle$		$AVar = \langle \text{“}Tom\text{”}, 1 \rangle$
$BVar = \langle \text{“}Ann\text{”}, 0 \rangle$	$\longrightarrow$	$BVar = \langle \text{“}Ann\text{”}, 0 \rangle$
$AtoB2 = \langle Bad, \langle \text{“}Tom\text{”}, 1 \rangle \rangle$		$AtoB = \langle \langle \text{“}Tom\text{”}, 1 \rangle \rangle$
$BtoA2 = \langle 0, Bad, 0, Bad \rangle$		$BtoA =$

For this state in a behavior of $AB2$ here's how we get the corresponding state in a behavior of AB.

The values of $AVar$ and $BVar$ are the same.

We obtain the sequence of messages $AtoB$ from the sequence of messages $AtoB2$ by deleting the Bad messages.

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

State of $AB2$		State of AB
$AVar = \langle \text{"}Tom\text{"}, 1 \rangle$		$AVar = \langle \text{"}Tom\text{"}, 1 \rangle$
$BVar = \langle \text{"}Ann\text{"}, 0 \rangle$	$\longrightarrow$	$BVar = \langle \text{"}Ann\text{"}, 0 \rangle$
$AtoB2 = \langle Bad, \langle \text{"}Tom\text{"}, 1 \rangle \rangle$		$AtoB = \langle \langle \text{"}Tom\text{"}, 1 \rangle \rangle$
$BtoA2 = \langle 0, Bad, 0, Bad \rangle$		$BtoA =$

And we do the same thing to obtain the sequence of messages $BtoA$ from the sequence of messages $BtoA2$.

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

State of $AB2$		State of AB
$AVar = \langle \text{“}Tom\text{”}, 1 \rangle$		$AVar = \langle \text{“}Tom\text{”}, 1 \rangle$
$BVar = \langle \text{“}Ann\text{”}, 0 \rangle$	$\longrightarrow$	$BVar = \langle \text{“}Ann\text{”}, 0 \rangle$
$AtoB2 = \langle Bad, \langle \text{“}Tom\text{”}, 1 \rangle \rangle$		$AtoB = \langle \langle \text{“}Tom\text{”}, 1 \rangle \rangle$
$BtoA2 = \langle 0, Bad, 0, Bad \rangle$		$BtoA = \langle 0, 0 \rangle$

And we do the same thing to obtain the sequence of messages $BtoA$ from the sequence of messages $BtoA2$.

The $AB2$ protocol implements the AB protocol, where $CorruptMsg$ steps implement $LoseMsg$ steps.

What does it mean?

For every behavior of $AB2$ we can obtain a behavior of AB by changing the state as follows:

State of $AB2$		State of AB
$AVar = \langle \text{"}Tom\text{"}, 1 \rangle$		$AVar = \langle \text{"}Tom\text{"}, 1 \rangle$
$BVar = \langle \text{"}Ann\text{"}, 0 \rangle$	$\longrightarrow$	$BVar = \langle \text{"}Ann\text{"}, 0 \rangle$
$AtoB2 = \langle Bad, \langle \text{"}Tom\text{"}, 1 \rangle \rangle$		$AtoB = \langle \langle \text{"}Tom\text{"}, 1 \rangle \rangle$
$BtoA2 = \langle 0, Bad, 0, Bad \rangle$		$BtoA = \langle 0, 0 \rangle$

And we do the same thing to obtain the sequence of messages $BtoA$ from the sequence of messages $BtoA2$.

State of $AB2$		State of AB
$\begin{aligned} AVar &= \langle \text{"}Tom\text{"}, 1\rangle \\ BVar &= \langle \text{"}Ann\text{"}, 0\rangle \\ AtoB2 &= \langle Bad, \langle \text{"}Tom\text{"}, 1\rangle\rangle \\ BtoA2 &= \langle 0, Bad, 0, Bad\rangle \end{aligned}$	$\longrightarrow$	$\begin{aligned} AVar &= \langle \text{"}Tom\text{"}, 1\rangle \\ BVar &= \langle \text{"}Ann\text{"}, 0\rangle \\ AtoB &= \langle\langle \text{"}Tom\text{"}, 1\rangle\rangle \\ BtoA &= \langle 0, 0\rangle \end{aligned}$

"$AB2$ implements AB" means that this transformation of states of the $AB2$ protocol to states of the AB protocol

State of $AB2$			State of AB
$AVar = \langle \text{“}Tom\text{”}, 1 \rangle$			$AVar = \langle \text{“}Tom\text{”}, 1 \rangle$
$BVar = \langle \text{“}Ann\text{”}, 0 \rangle$		$\longrightarrow$	$BVar = \langle \text{“}Ann\text{”}, 0 \rangle$
$AtoB2 = \langle Bad, \langle \text{“}Tom\text{”}, 1 \rangle \rangle$			$AtoB = \langle \langle \text{“}Tom\text{”}, 1 \rangle \rangle$
$BtoA2 = \langle 0, Bad, 0, Bad \rangle$			$BtoA = \langle 0, 0 \rangle$

Behavior of $AB2$ $\longrightarrow$ Behavior of AB

“$AB2$ implements AB” means that this transformation of states of the $AB2$ protocol to states of the AB protocol

transforms a behavior of the $AB2$ protocol to a behavior of the AB protocol.

State of $AB2$ → State of AB

$$\begin{array}{lcl} AVar & = & \langle \text{“}Tom\text{”}, 1\rangle \\ BVar & = & \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB2 & = & \langle Bad, \langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA2 & = & \langle 0, Bad, 0, Bad\rangle \end{array} \longrightarrow \begin{array}{lcl} AVar & = & \langle \text{“}Tom\text{”}, 1\rangle \\ BVar & = & \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB & = & \langle\langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA & = & \langle 0, 0\rangle \end{array}$$

Behavior of $AB2$ $\longrightarrow$ Behavior of AB

To show this implementation, we first transform states of the $AB2$ protocol to produce behaviors satisfying a new specification $SpecH$.

State of $AB2$

$$\begin{aligned} AVar &= \langle \text{“}Tom\text{”}, 1\rangle \\ BVar &= \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB2 &= \langle Bad, \langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA2 &= \langle 0, Bad, 0, Bad\rangle \end{aligned}$$

$\longrightarrow$

State of AB

$$\begin{aligned} AVar &= \langle \text{“}Tom\text{”}, 1\rangle \\ BVar &= \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB &= \langle\langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA &= \langle 0, 0\rangle \end{aligned}$$

State of $SpecH$

To show this implementation, we first transform states of the $AB2$ protocol to produce behaviors satisfying a new specification $SpecH$. We obtain a state of $SpecH$ by

State of $AB2$

$$\begin{array}{lcl} AVar & = & \langle \text{“}Tom\text{”}, 1\rangle \\ BVar & = & \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB2 & = & \langle Bad, \langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA2 & = & \langle 0, Bad, 0, Bad\rangle \end{array}$$

$\longrightarrow$

State of AB

$$\begin{array}{lcl} AVar & = & \langle \text{“}Tom\text{”}, 1\rangle \\ BVar & = & \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB & = & \langle\langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA & = & \langle 0, 0\rangle \end{array}$$

State of $SpecH$

$$\begin{array}{lcl} AVar & = & \langle \text{“}Tom\text{”}, 1\rangle \\ BVar & = & \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB2 & = & \langle Bad, \langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA2 & = & \langle 0, Bad, 0, Bad\rangle \end{array}$$

To show this implementation, we first transform states of the $AB2$ protocol to produce behaviors satisfying a new specification $SpecH$. We obtain a state of $SpecH$ by starting with a state of $AB2$

State of $AB2$

$$\begin{aligned} AVar &= \langle \text{"}Tom\text{"}, 1\rangle \\ BVar &= \langle \text{"}Ann\text{"}, 0\rangle \\ AtoB2 &= \langle Bad, \langle \text{"}Tom\text{"}, 1\rangle\rangle \\ BtoA2 &= \langle 0, Bad, 0, Bad\rangle \end{aligned}$$

$\longrightarrow$

State of AB

$$\begin{aligned} AVar &= \langle \text{"}Tom\text{"}, 1\rangle \\ BVar &= \langle \text{"}Ann\text{"}, 0\rangle \\ AtoB &= \langle\langle \text{"}Tom\text{"}, 1\rangle\rangle \\ BtoA &= \langle 0, 0\rangle \end{aligned}$$

State of $SpecH$

$$\begin{aligned} AVar &= \langle \text{"}Tom\text{"}, 1\rangle \\ BVar &= \langle \text{"}Ann\text{"}, 0\rangle \\ AtoB2 &= \langle Bad, \langle \text{"}Tom\text{"}, 1\rangle\rangle \\ BtoA2 &= \langle 0, Bad, 0, Bad\rangle \\ AtoB &= \langle\langle \text{"}Tom\text{"}, 1\rangle\rangle \\ BtoA &= \langle 0, 0\rangle \end{aligned}$$

To show this implementation, we first transform states of the $AB2$ protocol to produce behaviors satisfying a new specification $SpecH$. We obtain a state of $SpecH$ by starting with a state of $AB2$

and then adding the values of the variables $AtoB$ and $BtoA$ from the state of AB.

State of $AB2$

$$\begin{aligned} AVar &= \langle \text{“}Tom\text{”}, 1\rangle \\ BVar &= \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB2 &= \langle Bad, \langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA2 &= \langle 0, Bad, 0, Bad\rangle \end{aligned}$$

$\longrightarrow$

State of AB

$$\begin{aligned} AVar &= \langle \text{“}Tom\text{”}, 1\rangle \\ BVar &= \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB &= \langle\langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA &= \langle 0, 0\rangle \end{aligned}$$

State of $SpecH$

$$\begin{aligned} AVar &= \langle \text{“}Tom\text{”}, 1\rangle \\ BVar &= \langle \text{“}Ann\text{”}, 0\rangle \\ AtoB2 &= \langle Bad, \langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA2 &= \langle 0, Bad, 0, Bad\rangle \\ AtoB &= \langle\langle \text{“}Tom\text{”}, 1\rangle\rangle \\ BtoA &= \langle 0, 0\rangle \end{aligned}$$

To show this implementation, we first transform states of the $AB2$ protocol to produce behaviors satisfying a new specification $SpecH$. We obtain a state of $SpecH$ by starting with a state of $AB2$

and then adding the values of the variables $AtoB$ and $BtoA$ from the state of AB.

State of $SpecH$

$$
\begin{aligned}
AVar &= \langle \text{“}Tom\text{”}, 1\rangle \\
BVar &= \langle \text{“}Ann\text{”}, 0\rangle \\
AtoB2 &= \langle Bad, \langle \text{“}Tom\text{”}, 1\rangle\rangle \\
BtoA2 &= \langle 0, Bad, 0, Bad\rangle \\
AtoB &= \langle\langle \text{“}Tom\text{”}, 1\rangle\rangle \\
BtoA &= \langle 0, 0\rangle
\end{aligned}
$$

The $AB2$ protocol implements the AB protocol if and only if every behavior allowed by formula $SpecH$ is a behavior of the AB protocol.

State of $SpecH$

$$
\begin{aligned}
AVar &= \langle \text{“}Tom\text{”}, 1 \rangle \\
BVar &= \langle \text{“}Ann\text{”}, 0 \rangle \\
AtoB2 &= \langle Bad, \langle \text{“}Tom\text{”}, 1 \rangle \rangle \\
BtoA2 &= \langle 0, Bad, 0, Bad \rangle \\
AtoB &= \langle \langle \text{“}Tom\text{”}, 1 \rangle \rangle \\
BtoA &= \langle 0, 0 \rangle
\end{aligned}
$$

The $AB2$ protocol implements the AB protocol iff every behavior allowed by $SpecH$ is a behavior of the AB protocol.

The $AB2$ protocol implements the AB protocol if and only if every behavior allowed by formula $SpecH$ is a behavior of the AB protocol.

State of $SpecH$

$$
\begin{aligned}
AVar &= \langle \text{“}Tom\text{”}, 1 \rangle \\
BVar &= \langle \text{“}Ann\text{”}, 0 \rangle \\
AtoB2 &= \langle Bad, \langle \text{“}Tom\text{”}, 1 \rangle \rangle \\
BtoA2 &= \langle 0, Bad, 0, Bad \rangle \\
AtoB &= \langle \langle \text{“}Tom\text{”}, 1 \rangle \rangle \\
BtoA &= \langle 0, 0 \rangle
\end{aligned}
$$

The $AB2$ protocol implements the AB protocol iff every behavior allowed by $SpecH$ is a behavior of the AB protocol.

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

The $AB2$ protocol implements the AB protocol if and only if every behavior allowed by formula $SpecH$ is a behavior of the AB protocol.

This condition is expressed by the theorem that formula $SpecH$ implies formula $Spec$ of module AB.

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

This answers our first question: What does it mean for $AB2$ to implement AB?

THEOREM $SpecH \;\Rightarrow\;$ formula $Spec$ of module AB

This answers the first question:

What does it mean for $AB2$ to implement AB?

This answers our first question: What does it mean for $AB2$ to implement AB?

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

This answers the first question:

What does it mean for $AB2$ to implement AB?

We now answer the second question:

How do we check it?

This answers our first question: What does it mean for $AB2$ to implement AB?

We now answer the second question: How do we check it?

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

This answers the first question:

What does it mean for $AB2$ to implement AB ?

We now answer the second question:

How do we check it?

To do this, we first write $SpecH$ in TLA+.

This answers our first question: What does it mean for $AB2$ to implement AB ?

We now answer the second question: How do we check it?

To do this, we first actually write the formula $SpecH$ in TLA+.

SPECIFYING SpecH

A behavior should satisfy $SpecH$ iff:

A behavior should satisfy $SpecH$ if and only if the following conditions hold.

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.

A behavior should satisfy $SpecH$ if and only if the following conditions hold.

First, the values of the four variables of the $AB2$ spec should satisfy that spec.

A behavior should satisfy $SpecH$ iff:

– The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.

– In every state:

A behavior should satisfy $SpecH$ if and only if the following conditions hold.

First, the values of the four variables of the $AB2$ spec should satisfy that spec.

Second, in every state of the behavior,

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.

A behavior should satisfy $SpecH$ if and only if the following conditions hold.

First, the values of the four variables of the $AB2$ spec should satisfy that spec.

Second, in every state of the behavior, $AtoB$ should equal the sequence obtained from $AtoB2$ by removing corrupted messages.

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.
  - $BtoA = BtoA2$ without corrupted messages.

And $BtoA$ should equal the sequence obtained from $BtoA2$ by removing corrupted messages.

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.
  - $BtoA = BtoA2$ without corrupted messages.

$SpecH$ should equal

$\wedge$

$\wedge$

And $BtoA$ should equal the sequence obtained from $BtoA2$ by removing corrupted messages.

So, $SpecH$ should be the conjunction of two formulas.

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.
  - $BtoA = BtoA2$ without corrupted messages.

$SpecH$ should equal

$\land$

$\land$

And $BtoA$ should equal the sequence obtained from $BtoA2$ by removing corrupted messages.

So, $SpecH$ should be the conjunction of two formulas.

The first formula, which expresses this condition,

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.
  - $BtoA = BtoA2$ without corrupted messages.

$SpecH$ should equal

$\wedge$ Formula $Spec$ of module $AB2$

$\wedge$

And $BtoA$ should equal the sequence obtained from $BtoA2$ by removing corrupted messages.

So, $SpecH$ should be the conjunction of two formulas.

The first formula, which expresses this condition, is just formula $Spec$ of module $AB2$.

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.
  - $BtoA = BtoA2$ without corrupted messages.

$SpecH$ should equal

$\land$ Formula $Spec$ of module $AB2$

$\land$

And $BtoA$ should equal the sequence obtained from $BtoA2$ by removing corrupted messages.

So, $SpecH$ should be the conjunction of two formulas.

The first formula, which expresses this condition, is just formula $Spec$ of module $AB2$.

The second formula asserts that something is true in every state,

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.
  - $BtoA = BtoA2$ without corrupted messages.

$SpecH$ should equal

$\land$ Formula $Spec$ of module $AB2$

$\land$ $\Box$

which is expressed by the temporal operator $always$.

A behavior should satisfy $SpecH$ iff:

– The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.

– In every state:

  – $AtoB = AtoB2$ without corrupted messages.

  – $BtoA = BtoA2$ without corrupted messages.

$SpecH$ should equal

$\wedge$ Formula $Spec$ of module $AB2$

$\wedge$ $\Box$

which is expressed by the temporal operator $always$.

The condition satisfied by every state

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.
  - $BtoA = BtoA2$ without corrupted messages.

$SpecH$ should equal

$\land$ Formula $Spec$ of module $AB2$

$\land \; \Box \; \land \; AtoB = AtoB2$ without corrupted messages.

$\quad\quad\; \land \; BtoA = BtoA2$ without corrupted messages.

which is expressed by the temporal operator $always$.

The condition satisfied by every state
is the conjunction of these two conditions.

A behavior should satisfy $SpecH$ iff:

- The values of $AVar$, $BVar$, $AtoB2$, $BtoA2$ satisfy the $AB2$ spec.
- In every state:
  - $AtoB = AtoB2$ without corrupted messages.
  - $BtoA = BtoA2$ without corrupted messages.

$SpecH$ should equal

$\land$ Formula $Spec$ of module $AB2$

$\land$ $\Box$ $\land$ $AtoB = AtoB2$ without corrupted messages.
$\quad\quad$ $\land$ $BtoA = BtoA2$ without corrupted messages.

which is expressed by the temporal operator $always$.

The condition satisfied by every state
is the conjunction of these two conditions.

So here's what $SpecH$ should equal.

$$
\begin{aligned}
SpecH \;\triangleq\; &\land \text{ Formula } Spec \text{ of module } AB2\\
&\land \Box \land AtoB = AtoB2 \text{ without corrupted messages.}\\
& \land BtoA = BtoA2 \text{ without corrupted messages.}
\end{aligned}
$$

So the definition of $SpecH$ should look like this.

$$
\begin{aligned}
SpecH \;\triangleq\; & \land \text{ Formula } Spec \text{ of module } AB2 \\
& \land \Box \land AtoB = AtoB2 \text{ without corrupted messages.} \\
& \land BtoA = BtoA2 \text{ without corrupted messages.}
\end{aligned}
$$

$SpecH$ is defined in module $AB2H$.

So the definition of $SpecH$ should look like this.

We define $SpecH$ in another module called $AB2H$.

$SpecH \triangleq \land$ Formula $Spec$ of module $AB2$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

$SpecH$ is defined in module $AB2H$.

Stop the video and download that module now.

So the definition of $SpecH$ should look like this.

We define $SpecH$ in another module called $AB2H$.

Stop the video and download that module now.

$$SpecH \triangleq \land \text{ Formula } Spec \text{ of module } AB2$$
$$\quad\quad \land \Box \land AtoB = AtoB2 \text{ without corrupted messages.}$$
$$\quad\quad\quad\quad \land BtoA = BtoA2 \text{ without corrupted messages.}$$

We start by writing this conjunct.

$SpecH \triangleq \land$ Formula $Spec$ of module $AB2$
$\land \Box \land AtoB = AtoB2$ without corrupted messages.
$\land BtoA = BtoA2$ without corrupted messages.

To permit $AB2H$ to import $Spec$ from $AB2$

To permit module $AB2H$ to import formula $Spec$ from module $AB2$

$SpecH \triangleq$ $\land$ Formula $Spec$ of module $AB2$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

To permit $AB2H$ to import $Spec$ from $AB2$, it extends the same modules and declares the same constants and variables as $AB2$.

To permit module $AB2H$ to import formula $Spec$ from module $AB2$

$AB2H$ begins by extending the same modules and declaring the same constants and variables as module $AB2$.

$SpecH \triangleq \land$ Formula $Spec$ of module $AB2$
$\land \Box \land AtoB = AtoB2$ without corrupted messages.
$\land BtoA = BtoA2$ without corrupted messages.

To permit $AB2H$ to import $Spec$ from $AB2$, it extends the same modules and declares the same constants and variables as $AB2$.

EXTENDS $Integers, Sequences$

CONSTANTS $Data, Bad$

ASSUME $Bad \notin (Data \times \{0,1\}) \cup \{0,1\}$

VARIABLES $AVar, BVar, AtoB, BtoA$

To permit module $AB2H$ to import formula $Spec$ from module $AB2$

$AB2H$ begins by extending the same modules and declaring the same constants and variables as module $AB2$.

$$SpecH \triangleq \land \text{ Formula } Spec \text{ of module } AB2$$
$$\land \square \land AtoB = AtoB2 \text{ without corrupted messages.}$$
$$\land BtoA = BtoA2 \text{ without corrupted messages.}$$

$AB2 \triangleq$ INSTANCE $AB2$

The module next imports the definitions from module $AB2$ with this *instance* statement.

This imports

$SpecH \triangleq \land$ Formula $Spec$ of module $AB2$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

$AB2 \triangleq$ INSTANCE $AB2$

The module next imports the definitions from module $AB2$ with this *instance* statement.

This imports formula $Spec$ of module $AB2$ as

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = AtoB2 \text{ without corrupted messages.}$$
$$\land BtoA = BtoA2 \text{ without corrupted messages.}$$

$AB2 \triangleq$ INSTANCE $AB2$

The module next imports the definitions from module $AB2$ with this *instance* statement.

This imports formula $Spec$ of module $AB2$ as $AB2$ bang $Spec$.

$$
\begin{aligned}
SpecH \triangleq\ & \land\ AB2!Spec \\
& \land\ \Box \land\ AtoB = AtoB2 \text{ without corrupted messages.} \\
& \qquad\ \land\ BtoA = BtoA2 \text{ without corrupted messages.}
\end{aligned}
$$

$$AB2 \triangleq \text{INSTANCE } AB2$$

The module next imports the definitions from module $AB2$ with this *instance* statement.

This imports formula $Spec$ of module $AB2$ as $AB2$ bang $Spec$.

To write this part of the definition of $SpecH$,

$SpecH \triangleq \land AB2!Spec$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

$AB2 \triangleq$ INSTANCE $AB2$

VARIABLES $AtoB, BtoA$

The module next imports the definitions from module $AB2$ with this *instance* statement.

This imports formula $Spec$ of module $AB2$ as $AB2$ bang $Spec$.

To write this part of the definition of $SpecH$,

the module has to declare the variables $AtoB$ and $BtoA$.

$$
\begin{aligned}
SpecH \triangleq\ & \land\ AB2!Spec \\
& \land\ \Box \land AtoB = AtoB2 \text{ without corrupted messages.} \\
& \qquad \land BtoA = BtoA2 \text{ without corrupted messages.}
\end{aligned}
$$

$AB2 \triangleq$ INSTANCE $AB2$

VARIABLES $AtoB,\ BtoA$

To write this part of the definition,

$SpecH \triangleq \land AB2!Spec$
$\land \Box \land AtoB = AtoB2$ without corrupted messages.
$\land BtoA = BtoA2$ without corrupted messages.

$AB2 \triangleq$ INSTANCE $AB2$

VARIABLES $AtoB,\ BtoA$

Defines $RemoveBad(seq)$ to be the sequence obtained by removing Bad elements from a sequence seq .

To write this part of the definition,
the module next defines the operator $RemoveBad$ so that $RemoveBad$ of seq is the sequence obtained by removing elements equal to Bad from a sequence seq .

$SpecH \triangleq \land AB2!Spec$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

RECURSIVE $RemoveBad(\_)$

The definition of course is almost identical to the recursive definition of $RemoveX$ in part 1 of this lecture. It begins with a RECURSIVE declaration.

$SpecH \triangleq \land AB2!Spec$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

RECURSIVE $RemoveBad(\_)$

$RemoveBad(seq) \triangleq$

The definition of course is almost identical to the recursive definition of $RemoveX$ in part 1 of this lecture. It begins with a RECURSIVE declaration.

It then defines $RemoveBad$ of seq to be

$SpecH \triangleq \land AB2!Spec$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

RECURSIVE $RemoveBad(\_)$

$RemoveBad(seq) \triangleq$

IF $seq = \langle\,\rangle$

THEN $\langle\,\rangle$

ELSE

The definition of course is almost identical to the recursive definition of $RemoveX$ in part 1 of this lecture. It begins with a RECURSIVE declaration.

It then defines $RemoveBad$ of seq to be If seq is the empty sequence, then the empty sequence.

$SpecH \triangleq \land AB2!Spec$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

RECURSIVE $RemoveBad(\_)$

$RemoveBad(seq) \triangleq$

IF $seq = \langle \rangle$

THEN $\langle \rangle$

ELSE IF $Head(seq) = Bad$

THEN

ELSE

The definition of course is almost identical to the recursive definition of $RemoveX$ in part 1 of this lecture. It begins with a RECURSIVE declaration.

It then defines $RemoveBad$ of seq to be If seq is the empty sequence, then the empty sequence.

else if the head of seq equals Bad,

$SpecH \triangleq \land AB2!Spec$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

```
RECURSIVE RemoveBad(_)
RemoveBad(seq) ≜
    IF seq = ⟨ ⟩
      THEN ⟨ ⟩
      ELSE IF Head(seq) = Bad
              THEN RemoveBad(Tail(seq))
              ELSE
```

The definition of course is almost identical to the recursive definition of $RemoveX$ in part 1 of this lecture. It begins with a RECURSIVE declaration.

It then defines $RemoveBad$ of seq to be If seq is the empty sequence, then the empty sequence.

else if the head of seq equals Bad, then $RemoveBad$ of the tail of seq.

[slide 75]

$SpecH \triangleq \land AB2!Spec$

$\land \Box \land AtoB = AtoB2$ without corrupted messages.

$\land BtoA = BtoA2$ without corrupted messages.

RECURSIVE $RemoveBad(\_)$

$RemoveBad(seq) \triangleq$

IF $seq = \langle\, \rangle$

THEN $\langle\, \rangle$

ELSE IF $Head(seq) = Bad$

THEN $RemoveBad(Tail(seq))$

ELSE $\langle Head(seq) \rangle \circ RemoveBad(Tail(seq))$

Else, the sequence obtained by prepending the head of seq to the front of $RemoveBad$ of the tail of seq.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = AtoB2 \text{ without corrupted messages.}$$
$$\land BtoA = BtoA2 \text{ without corrupted messages.}$$

```
RECURSIVE RemoveBad(_)
RemoveBad(seq) ≜
    IF seq = ⟨ ⟩
      THEN ⟨ ⟩
      ELSE IF Head(seq) = Bad
               THEN RemoveBad(Tail(seq))
               ELSE ⟨Head(seq)⟩ ∘ RemoveBad(Tail(seq))
```

Else, the sequence obtained by prepending the head of seq to the front of $RemoveBad$ of the tail of seq.

We can use it to replace these pseudo-expressions

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

RECURSIVE $RemoveBad(\_)$

$$RemoveBad(seq) \triangleq$$
$$\text{IF } seq = \langle\,\rangle$$
$$\text{THEN } \langle\,\rangle$$
$$\text{ELSE IF } Head(seq) = Bad$$
$$\text{THEN } RemoveBad(Tail(seq))$$
$$\text{ELSE } \langle Head(seq)\rangle \circ RemoveBad(Tail(seq))$$

Else, the sequence obtained by prepending the head of seq to the front of $RemoveBad$ of the tail of seq.

We can use it to replace these pseudo-expressions with real expressions.

$$
\begin{aligned}
SpecH \triangleq\ & \land\ AB2!Spec \\
& \land\ \Box \land AtoB = RemoveBad(AtoB2) \\
& \land BtoA = RemoveBad(BtoA2)
\end{aligned}
$$

Else, the sequence obtained by prepending the head of seq to the front of $RemoveBad$ of the tail of seq.

We can use it to replace these pseudo-expressions with real expressions.

This completes the definition of $SpecH$, which comes next in the module.

$$
\begin{aligned}
SpecH \;\triangleq\; & \land\ AB2!Spec \\
& \land\ \Box \land\ AtoB\ =\ RemoveBad(AtoB2) \\
& \land\ BtoA\ =\ RemoveBad(BtoA2)
\end{aligned}
$$

$AtoB$ and $BtoA$ are imaginary variables

$AtoB$ and $BtoA$ are imaginary variables

$$
\begin{aligned}
SpecH \triangleq\ & \land\ AB2!Spec \\
& \land\ \Box \land AtoB = RemoveBad(AtoB2) \\
& \land BtoA = RemoveBad(BtoA2)
\end{aligned}
$$

$AtoB$ and $BtoA$ are imaginary variables added to $AB2!Spec$ to show that it implements the AB protocol spec.

$AtoB$ and $BtoA$ are imaginary variables added to $AB2!Spec$ to show that it implements the AB protocol spec.

$$SpecH \triangleq \wedge\ AB2!Spec$$
$$\wedge\ \Box \wedge\ AtoB = RemoveBad(AtoB2)$$
$$\wedge\ BtoA = RemoveBad(BtoA2)$$

$AtoB$ and $BtoA$ are imaginary variables added to $AB2!Spec$ to show that it implements the AB protocol spec.

They are not meant to be implemented by the $AB2$ protocol.

$AtoB$ and $BtoA$ are imaginary variables added to $AB2!Spec$ to show that it implements the AB protocol spec.

They are not meant to be implemented by the $AB2$ protocol.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$AtoB$ and $BtoA$ are imaginary variables added to $AB2!Spec$ to show that it implements the AB protocol spec.

They are not meant to be implemented by the $AB2$ protocol.

If we ignore the values of $AtoB$ and $BtoA$, then $SpecH$ and $AB2!Spec$ allow the same behaviors.

$AtoB$ and $BtoA$ are imaginary variables added to $AB2!Spec$ to show that it implements the AB protocol spec.

They are not meant to be implemented by the $AB2$ protocol.

If we ignore the values of $AtoB$ and $BtoA$, then $SpecH$ and $AB2!Spec$ allow the same behaviors.

CHECKING IMPLEMENTATION

Our goal is to check:

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

Remember that our goal is to check this theorem

Our goal is to check:

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

which asserts that $AB2$ implements AB.

Remember that our goal is to check this theorem which asserts that the $AB2$ protocol implements the AB protocol.

But first we have to write it in TLA+.

Our goal is to check:

THEOREM $\boxed{SpecH} \Rightarrow$ formula $Spec$ of module AB

This is defined in $AB2H$.

Remember that our goal is to check this theorem which asserts that the $AB2$ protocol implements the AB protocol.

But first we have to write it in TLA+.

We just defined $SpecH$ in module $AB2H$.

Our goal is to check:

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

Have to write in $AB2H$.

Remember that our goal is to check this theorem which asserts that the $AB2$ protocol implements the AB protocol.

But first we have to write it in TLA+.

We just defined $SpecH$ in module $AB2H$.

We now have to write formula $Spec$ of module AB in module $AB2H$. But that's easy.

Our goal is to check:

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

$AB \triangleq$ INSTANCE AB

We just add this INSTANCE statement to module $AB2H$.

Our goal is to check:

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

$AB \triangleq$ INSTANCE AB

THEOREM $SpecH \Rightarrow AB!Spec$

We just add this INSTANCE statement to module $AB2H$.

which defines AB bang $Spec$ to be this formula.

Our goal is to check:

THEOREM $SpecH \Rightarrow$ formula $Spec$ of module AB

$AB \triangleq$ INSTANCE AB

THEOREM $SpecH \Rightarrow AB!Spec$

We just add this INSTANCE statement to module $AB2H$.

which defines AB bang $Spec$ to be this formula.

THEOREM $SpecH \Rightarrow AB!Spec$

TLC can't check this theorem

THEOREM $SpecH \Rightarrow AB!Spec$

TLC can't check this theorem

THEOREM $SpecH \Rightarrow AB!Spec$

TLC can’t check this theorem

TLC can’t check this theorem

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

THEOREM $SpecH \Rightarrow AB!Spec$

TLC can't check this theorem because $SpecH$

TLC can't check this theorem

because $SpecH$

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

THEOREM $SpecH \Rightarrow AB!Spec$

TLC can't check this theorem because $SpecH$ doesn't have the standard form for a TLA+ safety spec:

$$InitH \land \Box[NextH]_{varsH}$$

TLC can't check this theorem

because $SpecH$ doesn't have the standard form for a TLA+ safety spec — which has an initial-state formula and a next-state action.

We could solve this problem be rewriting $SpecH$.

We could solve this problem be rewriting $SpecH$.

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining a specification $SpecHH$ that TLC can handle

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining a specification $SpecHH$ that TLC can handle

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining a specification $SpecHH$ that TLC can handle and is equivalent to $SpecH$.

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining a specification $SpecHH$ that TLC can handle and is equivalent to $SpecH$.

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining
a specification $SpecHH$ that TLC can handle
and is equivalent to $SpecH$.

You can read module $AB2H$ to see how it's done.

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining a specification $SpecHH$ that TLC can handle and is equivalent to $SpecH$.

You can read module $AB2H$ to see how it's done.

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining
a specification $SpecHH$ that TLC can handle
and is equivalent to $SpecH$.

You can read module $AB2H$ to see how it's done.

I'll take a longer approach

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining a specification $SpecHH$ that TLC can handle and is equivalent to $SpecH$.

You can read module $AB2H$ to see how it's done.

Here, I'll take a longer approach

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining
a specification $SpecHH$ that TLC can handle
and is equivalent to $SpecH$.

You can read module $AB2H$ to see how it's done.

I'll take a longer approach that leads to greater insight
into implementation.

We could solve this problem be rewriting $SpecH$.

This is done in module $AB2H$ by defining a specification $SpecHH$ that TLC can handle and is equivalent to $SpecH$.

You can read module $AB2H$ to see how it's done.

Here, I'll take a longer approach that leads to greater insight into implementation.

SIMPLIFYING REFINEMENT

$$
\begin{aligned}
SpecH \triangleq\ & \land\ AB2!Spec \\
& \land\ \Box \land AtoB = RemoveBad(AtoB2) \\
& \qquad \land BtoA = RemoveBad(BtoA2)
\end{aligned}
$$

$AB \triangleq$ INSTANCE AB

THEOREM $SpecH \Rightarrow AB!Spec$

This is where we are in Module $AB2H$.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

By the Temporal Substitution Rule

This is where we are in Module $AB2H$.

By the Temporal Substitution Rule

$$
\begin{aligned}
SpecH \;\triangleq\; & \land\; AB2!Spec \\
& \land\; \Box \land AtoB = RemoveBad(AtoB2) \\
& \qquad \land BtoA = RemoveBad(BtoA2)
\end{aligned}
$$

$AB \;\triangleq\;$ INSTANCE AB

THEOREM $SpecH \;\Rightarrow\; AB!Spec$

By the Temporal Substitution Rule, this formula implies

This is where we are in Module $AB2H$.

By the Temporal Substitution Rule this *always* formula implies

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$AB \triangleq$ INSTANCE AB

THEOREM $SpecH \Rightarrow AB!Spec$

By the Temporal Substitution Rule, this formula implies

$$AB!Spec = (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2),$$
$$BtoA \leftarrow RemoveBad(BtoA2))$$

This is where we are in Module $AB2H$.

By the Temporal Substitution Rule this *always* formula implies that AB bang $Spec$ equals AB bang $Spec$ with this substitution.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$SpecH$ implies

$$AB!Spec = (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), \; BtoA \leftarrow RemoveBad(BtoA2))$$

And since $SpecH$ implies the *always* formula, it also implies this equality.

$$SpecH \triangleq \land\ AB2!Spec$$
$$\land\ \Box \land\ AtoB = RemoveBad(AtoB2)$$
$$\land\ BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$SpecH$ implies

$$AB!Spec = (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

And since $SpecH$ implies the *always* formula, it also implies this equality.

Therefore, this theorem

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$$\text{THEOREM } SpecH \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

$SpecH$ implies

$$AB!Spec = (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

And since $SpecH$ implies the *always* formula, it also implies this equality.

Therefore, this theorem is equivalent to the theorem we get by replacing AB bang $Spec$ by AB bang $Spec$ with the substitutions.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$$\text{THEOREM } SpecH \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

And since $SpecH$ implies the *always* formula, it also implies this equality.

Therefore, this theorem is equivalent to the theorem we get by replacing AB bang $Spec$ by AB bang $Spec$ with the substitutions.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$AB \triangleq$ INSTANCE AB

THEOREM $SpecH \Rightarrow AB!Spec$

THEOREM $SpecH \Rightarrow$ ($AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2)$,
$BtoA \leftarrow RemoveBad(BtoA2)$)

Does not contain $AtoB$ or $BtoA$.

Since this formula is obtained by substituting for $AtoB$ and $BtoA$, it does not contain those two variables.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$AB \triangleq$ INSTANCE AB

THEOREM $SpecH \Rightarrow AB!Spec$

THEOREM $SpecH \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2))$

Does not contain $AtoB$ or $BtoA$.

Since this formula is obtained by substituting for $AtoB$ and $BtoA$, it does not contain those two variables.

Hence, in the theorem,

$$SpecH \triangleq \land\ AB2!Spec$$
$$\land\ \Box \land\ AtoB = RemoveBad(AtoB2)$$
$$\land\ BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$$\text{THEOREM } SpecH \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

Does not contain $AtoB$ or $BtoA$.

Since this formula is obtained by substituting for $AtoB$ and $BtoA$, it does not contain those two variables.

Hence, in the theorem, this *always* conjunct of $SpecH$ is irrelevant.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$AB \triangleq$ INSTANCE AB

THEOREM $SpecH \Rightarrow AB!Spec$

THEOREM $\boxed{SpecH} \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)\ )$

Since this formula is obtained by substituting for $AtoB$ and $BtoA$, it does not contain those two variables.

Hence, in the theorem, this *always* conjunct of $SpecH$ is irrelevant.

So we can replace $SpecH$ in the theorem

$$SpecH \triangleq \land \boxed{AB2!Spec}$$
$$\quad \land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\quad\quad \land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$$\text{THEOREM } \boxed{SpecH} \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

Since this formula is obtained by substituting for $AtoB$ and $BtoA$, it does not contain those two variables.

Hence, in the theorem, this *always* conjunct of $SpecH$ is irrelevant.

So we can replace $SpecH$ in the theorem by just this conjunct

$$SpecH \triangleq \land \boxed{AB2!Spec}$$
$$\quad \land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\quad\quad \land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$$\text{THEOREM } \boxed{SpecH} \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

$$\text{THEOREM } AB2!Spec \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

Since this formula is obtained by substituting for $AtoB$ and $BtoA$, it does not contain those two variables.

Hence, in the theorem, this *always* conjunct of $SpecH$ is irrelevant.

So we can replace $SpecH$ in the theorem by just this conjunct to get this equivalent theorem.

$$SpecH \triangleq \land AB2!Spec$$
$$\quad \land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\quad\quad \land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$$\text{THEOREM } SpecH \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

$$\text{THEOREM } AB2!Spec \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

Since this formula is obtained by substituting for $AtoB$ and $BtoA$, it does not contain those two variables.

Hence, in the theorem, this *always* conjunct of $SpecH$ is irrelevant.

So we can replace $SpecH$ in the theorem by just this conjunct to get this equivalent theorem.

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } SpecH \Rightarrow AB!Spec$$

$$\text{THEOREM } SpecH \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

$$\text{THEOREM } AB2!Spec \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), BtoA \leftarrow RemoveBad(BtoA2))$$

So we can replace the theorem we want to prove

$$SpecH \triangleq \land AB2!Spec$$
$$\land \Box \land AtoB = RemoveBad(AtoB2)$$
$$\land BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } AB2!Spec \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2),$$
$$BtoA \leftarrow RemoveBad(BtoA2))$$

So we can replace the theorem we want to prove

by this one.

$$
\begin{aligned}
SpecH \triangleq\ & \land\ AB2!Spec \\
& \land\ \Box \land AtoB = RemoveBad(AtoB2) \\
& \qquad \land BtoA = RemoveBad(BtoA2)
\end{aligned}
$$

$AB \triangleq$ INSTANCE AB

THEOREM $AB2!Spec \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)\ )$

And we can just check this theorem.

$$SpecH \triangleq \wedge\ AB2!Spec \\ \wedge\ \Box \wedge\ AtoB = RemoveBad(AtoB2) \\ \wedge\ BtoA = RemoveBad(BtoA2)$$

$$AB \triangleq \text{INSTANCE } AB$$

$$\text{THEOREM } AB2!Spec \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), \\ BtoA \leftarrow RemoveBad(BtoA2)\ )$$

And we can just check this theorem.

First, notice that $SpecH$ doesn't appear in the theorem any more, so we don't need to define it.

$AB \triangleq$ INSTANCE AB

THEOREM $AB2!Spec \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)\ )$

And we can just check this theorem.

First, notice that $SpecH$ doesn't appear in the theorem any more, so we don't need to define it.

These statements are in module $AB2H$.

$AB \triangleq$ INSTANCE AB

$$\text{THEOREM } AB2!Spec \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), \\ BtoA \leftarrow RemoveBad(BtoA2))$$

And we can just check this theorem.

First, notice that $SpecH$ doesn't appear in the theorem any more, so we don't need to define it.

The instance statement and the theorem are in module $AB2H$.

These statements are in module $AB2H$.

Let's move them to module $AB2$.

$AB \triangleq$ INSTANCE AB

THEOREM $AB2!Spec \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)\ )$

And we can just check this theorem.

First, notice that $SpecH$ doesn't appear in the theorem any more, so we don't need to define it.

The instance statement and the theorem are in module $AB2H$.

Let's move them to module $AB2$.

These statements are in module $AB2H$.

Let's move them to module $AB2$.

$AB \triangleq$ INSTANCE AB

$$\text{THEOREM } \boxed{AB2!Spec} \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), \; BtoA \leftarrow RemoveBad(BtoA2))$$

When we do that, the formula called $AB2!Spec$ in module $AB2H$

These statements are in module $AB2H$.

Let's move them to module $AB2$.

$AB \triangleq$ INSTANCE AB

THEOREM $Spec \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)\ )$

When we do that, the formula called $AB2!Spec$ in module $AB2H$ is simply called $Spec$.

These statements are in module $AB2H$.

Let's move them to module $AB2$.

$AB \triangleq$ INSTANCE AB

THEOREM $Spec \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)\ )$

When we do that, the formula called $AB2!Spec$ in module $AB2H$ is simply called $Spec$.

These statements are in module $AB2H$.

Let's move them to module $AB2$.

$AB \triangleq$ INSTANCE AB

THEOREM $\boxed{Spec} \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2))$

TLC can handle this specification.

When we do that, the formula called $AB2!Spec$ in module $AB2H$ is simply called $Spec$.

Inside module $AB2$, $Spec$ is an ordinary specification that TLC can handle.

These statements are in module $AB2H$.

Let's move them to module $AB2$.

$AB \triangleq$ INSTANCE AB

THEOREM $Spec \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)\ )$

But this isn't TLA+.

When we do that, the formula called $AB2!Spec$ in module $AB2H$ is simply called $Spec$.

Inside module $AB2$, $Spec$ is an ordinary specification that TLC can handle.

But this WITH formula is just a notation that I'm using here. It's not legal TLA+.

To see how to write it in TLA+,

These statements are in module $AB2H$.

Let's move them to module $AB2$.

$AB \triangleq$ INSTANCE AB

THEOREM $Spec \Rightarrow (AB!Spec$ WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)\ )$

When we do that, the formula called $AB2!Spec$ in module $AB2H$ is simply called $Spec$.

Inside module $AB2$, $Spec$ is an ordinary specification that TLC can handle.

But this WITH formula is just a notation that I'm using here. It's not legal TLA+.

To see how to write it in TLA+, we need to examine the INSTANCE statement.

$AB \triangleq$ INSTANCE AB

After expanding all definitions,

$AB \triangleq$ INSTANCE AB

After expanding all definitions,

After expanding all definitions,

$AB \triangleq$ INSTANCE AB

After expanding all definitions, formula $Spec$ of AB contains only TLA+ operators

After expanding all definitions, formula $Spec$ of module AB contains only TLA+ operators

$AB \triangleq$ INSTANCE AB

After expanding all definitions, formula $Spec$ of AB contains only TLA$^+$ operators and the declared symbols of AB :

After expanding all definitions, formula $Spec$ of module AB contains only TLA$^+$ operators and the declared symbols of the module, which are:

$AB \triangleq$ INSTANCE AB

After expanding all definitions, formula $Spec$ of AB contains only TLA$^+$ operators and the declared symbols of AB :

$Data,$

After expanding all definitions, formula $Spec$ of module AB contains only TLA$^+$ operators and the declared symbols of the module, which are: The constant $Data$

$AB \triangleq$ INSTANCE AB

After expanding all definitions, formula $Spec$ of AB contains only TLA$^+$ operators and the declared symbols of AB :

$Data, \ AVar, \ BVar, \ AtoB, \ BtoA$

After expanding all definitions, formula $Spec$ of module AB contains only TLA$^+$ operators and the declared symbols of the module, which are: The constant $Data$ and the module's four variables.

MODULE M

⋮

$AB \triangleq$ INSTANCE AB

After expanding all definitions, formula $Spec$ of AB contains only TLA$^+$ operators and the declared symbols of AB :

$Data,\ AVar,\ BVar,\ AtoB,\ BtoA$

To import a definition from AB into an arbitrary module M ,

After expanding all definitions, formula $Spec$ of module AB contains only TLA$^+$ operators and the declared symbols of the module, which are: The constant $Data$ and the module's four variables.

To import a definition from module AB into an arbitrary module M,

MODULE M

⋮

$AB \triangleq$ INSTANCE AB

After expanding all definitions, formula $Spec$ of AB contains only TLA$^+$ operators and the declared symbols of AB :

$Data,\ AVar,\ BVar,\ AtoB,\ BtoA$

To import a definition from AB into an arbitrary module M , we must substitute expressions of M for those symbols.

After expanding all definitions, formula $Spec$ of module AB contains only TLA$^+$ operators and the declared symbols of the module, which are: The constant $Data$ and the module's four variables.

To import a definition from module AB into an arbitrary module M, we must substitute expressions of module M for those symbols.

MODULE M

⋮

$AB \triangleq$ INSTANCE AB WITH

After expanding all definitions, formula $Spec$ of AB contains only TLA^{+} operators and the declared symbols of AB :

$Data,\ AVar,\ BVar,\ AtoB,\ BtoA$

To import a definition from AB into an arbitrary module M , we must substitute expressions of M for those symbols.

This is done by a WITH clause.

This is done by a WITH clause

MODULE M

⋮

$$
\begin{aligned}
AB \triangleq \text{INSTANCE } AB \text{ WITH } & Data \leftarrow \ldots, \\
& AVar \leftarrow \ldots, \quad BVar \leftarrow \ldots, \\
& AtoB \leftarrow \ldots, \quad BtoA \leftarrow \ldots
\end{aligned}
$$

This is done by a WITH clause

having this syntax,

MODULE M

⋮

$AB \triangleq$ INSTANCE AB WITH $Data \leftarrow \ldots,$
$AVar \leftarrow \ldots,\quad BVar \leftarrow \ldots,$
$AtoB \leftarrow \ldots,\quad BtoA \leftarrow \ldots$

The declared constants and variables of AB.

This is done by a WITH clause

having this syntax,

where these are the declared constants and variables of the instantiated module AB.

MODULE M

⋮

$$
\begin{aligned}
AB \triangleq \text{INSTANCE } AB \text{ WITH } & Data \leftarrow \boxed{\ldots}, \\
& AVar \leftarrow \boxed{\ldots}, \quad BVar \leftarrow \boxed{\ldots}, \\
& AtoB \leftarrow \boxed{\ldots}, \quad BtoA \leftarrow \boxed{\ldots}
\end{aligned}
$$

The declared constants and variables of AB.

The expressions of module M to be substituted for them.

This is done by a WITH clause

having this syntax,
where these are the declared constants and variables of the instantiated module AB.

And these are the expressions of the current module M to be substituted for them.

MODULE M

⋮

$AB \triangleq$ INSTANCE AB WITH $Data \leftarrow \ldots,$
$AVar \leftarrow \ldots, \quad BVar \leftarrow \ldots,$
$AtoB \leftarrow \ldots, \quad BtoA \leftarrow \ldots$

This is done by a WITH clause

having this syntax,
where these are the declared constants and variables of the instantiated module AB.

And these are the expressions of the current module M to be substituted for them.

MODULE $AB2H$

$\vdots$

$$
\begin{aligned}
AB \triangleq \text{INSTANCE } AB \text{ WITH } & Data \leftarrow \ldots, \\
& AVar \leftarrow \ldots, \quad BVar \leftarrow \ldots, \\
& AtoB \leftarrow \ldots, \quad BtoA \leftarrow \ldots
\end{aligned}
$$

When we instantiated module AB in module $AB2H$,

MODULE $AB2H$

$$
\begin{aligned}
AB \triangleq \text{INSTANCE } AB \text{ WITH } & Data \leftarrow \ldots, \\
& AVar \leftarrow \ldots, \quad BVar \leftarrow \ldots, \\
& AtoB \leftarrow \ldots, \quad BtoA \leftarrow \ldots
\end{aligned}
$$

When we instantiated module AB in module $AB2H$, for each of these declared symbols of module AB

MODULE $AB2H$

⋮

$$
\begin{aligned}
AB \triangleq \text{INSTANCE } AB \text{ WITH } & Data \leftarrow \boxed{Data}, \\
& AVar \leftarrow \boxed{AVar}, \quad BVar \leftarrow \boxed{BVar}, \\
& AtoB \leftarrow \boxed{AtoB}, \quad BtoA \leftarrow \boxed{BtoA}
\end{aligned}
$$

When we instantiated module AB in module $AB2H$, for each of these declared symbols of module AB we substituted the symbols of the same name from module $AB2H$.

MODULE $AB2H$

⋮

$AB \triangleq$ INSTANCE AB WITH $Data \leftarrow Data$,
$AVar \leftarrow AVar$, $BVar \leftarrow BVar$,
$AtoB \leftarrow AtoB$, $BtoA \leftarrow BtoA$

This is the default

When we instantiated module AB in module $AB2H$, for each of these declared symbols of module AB we substituted the symbols of the same name from module $AB2H$.

Substituting a symbol of the same name for a symbol of the instantiated module is the default

MODULE $AB2H$

⋮

$$AB \triangleq \text{INSTANCE } AB \text{ WITH } Data \leftarrow Data,\ AVar \leftarrow AVar,\ AtoB \leftarrow AtoB,\ BtoA \leftarrow BtoA$$

This is the default if we omit a substitution from the WITH clause.

When we instantiated module AB in module $AB2H$, for each of these declared symbols of module AB we substituted the symbols of the same name from module $AB2H$.

Substituting a symbol of the same name for a symbol of the instantiated module is the default
if we omit a substitution for that symbol from the WITH clause.

MODULE $AB2H$

⋮

$AB \triangleq$ INSTANCE AB

This is the default if we omit a substitution from the WITH clause.

So we could eliminate the WITH clause in module $AB2H$.

So we could eliminate the entire WITH clause from the INSTANCE statement in module $AB2H$.

MODULE $AB2H$

⋮

$AB \triangleq$ INSTANCE AB WITH $Data \leftarrow Data,$
$AVar \leftarrow AVar,\ BVar \leftarrow BVar,$
$AtoB \leftarrow AtoB,\ BtoA \leftarrow BtoA$

In module $AB2$

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $Data \leftarrow Data$,
$AVar \leftarrow AVar,\ BVar \leftarrow BVar,$
$AtoB \leftarrow AtoB,\ BtoA \leftarrow BtoA$

In module $AB2$

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $Data \leftarrow Data,$
$AVar \leftarrow AVar,\ BVar \leftarrow BVar,$
$AtoB \leftarrow AtoB,\ BtoA \leftarrow BtoA$

In module $AB2$ we want the default substitutions for $Data$, $AVar$, and $BVar$,

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH

$$AtoB \leftarrow AtoB, \; BtoA \leftarrow BtoA$$

In module $AB2$ we want the default substitutions for $Data$, $AVar$, and $BVar$, so we can omit them from the WITH clause.

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow AtoB,\ BtoA \leftarrow BtoA$

In module $AB2$ we want the default substitutions for $Data$, $AVar$, and $BVar$, so we can omit them from the WITH clause.

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow$ ~~$AtoB$~~, $BtoA \leftarrow$ ~~$BtoA$~~

$AtoB$ and $BtoA$ are undefined in $AB2$.

The symbols $AtoB$ and $BtoA$ are not declared in module $AB2$, so we need to substitute for them some expressions we can write in $AB2$.

If you remember how we got to this point, you should be able to guess that we're going to substitute

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow RemoveBad(AtoB2)$,
$BtoA \leftarrow RemoveBad(BtoA2)$

The symbols $AtoB$ and $BtoA$ are not declared in module $AB2$, so we need to substitute for them some expressions we can write in $AB2$.

If you remember how we got to this point, you should be able to guess that we're going to substitute these expressions for them – after adding the definition of $RemoveBad$ to module $AB2$.

MODULE $AB2$

$\vdots$

$$AB \triangleq \text{INSTANCE } AB \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), \quad BtoA \leftarrow RemoveBad(BtoA2)$$

$$\text{THEOREM } Spec \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), \quad BtoA \leftarrow RemoveBad(BtoA2)\,)$$

Recall that we were trying to check this theorem

MODULE $AB2$

$\vdots$

$$AB \triangleq \text{INSTANCE } AB \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), \quad BtoA \leftarrow RemoveBad(BtoA2)$$

$$\text{THEOREM } Spec \Rightarrow (AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2), \quad BtoA \leftarrow RemoveBad(BtoA2))$$

Recall that we were trying to check this theorem and we faced the problem that this isn't a TLA+ formula.

MODULE $AB2$

⋮

$$AB \triangleq \text{INSTANCE } AB \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2),\; BtoA \leftarrow RemoveBad(BtoA2)$$

$$\text{THEOREM } Spec \Rightarrow \boxed{(AB!Spec \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2),\; BtoA \leftarrow RemoveBad(BtoA2))}$$

This non-TLA+ formula

Recall that we were trying to check this theorem and we faced the problem that this isn't a TLA+ formula.

But this non-TLA+ formula

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow RemoveBad(AtoB2)$,
$BtoA \leftarrow RemoveBad(BtoA2)$

THEOREM $Spec \Rightarrow AB!Spec$ ~~WITH $AtoB \leftarrow RemoveBad(AtoB2)$, $BtoA \leftarrow RemoveBad(BtoA2)$~~

This non-TLA$^+$ formula can now be written as $AB!Spec$.

Recall that we were trying to check this theorem and we faced the problem that this isn't a TLA$^+$ formula.

But this non-TLA$^+$ formula can now be written simply as AB bang $Spec$

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow RemoveBad(AtoB2),$
$BtoA \leftarrow RemoveBad(BtoA2)$

THEOREM $Spec \Rightarrow AB!Spec$ ~~WITH $AtoB \leftarrow RemoveBad(AtoB2),$ $BtoA \leftarrow RemoveBad(BtoA2)$~~

This non-TLA$^+$ formula can now be written as $AB!Spec$.

The substitution is done by the INSTANCE statement.

Recall that we were trying to check this theorem and we faced the problem that this isn't a TLA$^+$ formula.

But this non-TLA$^+$ formula can now be written simply as AB bang $Spec$

Because the substitutions we wanted the formula to express are performed by the INSTANCE statement.

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow RemoveBad(AtoB2)$,
$BtoA \leftarrow RemoveBad(BtoA2)$

THEOREM $Spec \Rightarrow AB!Spec$

Recall that we were trying to check this theorem and we faced the problem that this isn't a TLA$^+$ formula.

But this non-TLA$^+$ formula can now be written simply as AB bang $Spec$

Because the substitutions we wanted the formula to express are performed by the INSTANCE statement.

MODULE $AB2$

⋮

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow RemoveBad(AtoB2)$,
$BtoA \leftarrow RemoveBad(BtoA2)$

THEOREM $Spec \Rightarrow AB!Spec$

TLC can check this theorem.

And TLC can now check this theorem.

Whew! We've finally reached our goal. But it took us so long, you may have forgotten why we wanted to get here. So, let's review what we've accomplished.

WHAT WE DID AND WHY

The $AB2$ protocol implements the AB protocol, where $RemoveBad(AtoB2)$ implements $AtoB$ and $RemoveBad(BtoA2)$ implements $BtoA$.

We saw that the $AB2$ protocol implements the AB protocol, where $RemoveBad$ of $AtoB2$ implements variable $AtoB$ of AB, and $RemoveBad$ of $BtoA2$ implements variable $BtoA$ of AB.

The $AB2$ protocol implements the AB protocol, where $RemoveBad(AtoB2)$ implements $AtoB$ and $RemoveBad(BtoA2)$ implements $BtoA$.

means

We saw that the $AB2$ protocol implements the AB protocol, where $RemoveBad$ of $AtoB2$ implements variable $AtoB$ of AB, and $RemoveBad$ of $BtoA2$ implements variable $BtoA$ of AB.

We then saw that this means that

The $AB2$ protocol implements the AB protocol, where $RemoveBad(AtoB2)$ implements $AtoB$ and $RemoveBad(BtoA2)$ implements $BtoA$.

means

$$\text{THEOREM} \quad \left(\begin{array}{l} \land\ Spec \text{ of } AB2 \\ \land\ \Box \land\ AtoB = RemoveBad(AtoB2) \\ \qquad \land\ BtoA = RemoveBad(BtoA2) \end{array}\right) \Rightarrow Spec \text{ of } AB$$

We saw that the $AB2$ protocol implements the AB protocol, where $RemoveBad$ of $AtoB2$ implements variable $AtoB$ of AB, and $RemoveBad$ of $BtoA2$ implements variable $BtoA$ of AB.

We then saw that this means that this theorem is true, The $AB2$ protocol implements the AB protocol, where

$RemoveBad(AtoB2)$ implements $AtoB$ and

$RemoveBad(BtoA2)$ implements $BtoA$.

means

$$\text{THEOREM}\ \left(\underbrace{\begin{array}{l}\land\ Spec\text{ of }AB2\\ \land\ \Box\ \land\ AtoB = RemoveBad(AtoB2)\\ \qquad\land\ BtoA = RemoveBad(BtoA2)\end{array}}_{SpecH}\right) \Rightarrow\ Spec\text{ of }AB$$

We saw that the $AB2$ protocol implements the AB protocol, where $RemoveBad$ of $AtoB2$ implements variable $AtoB$ of AB, and $RemoveBad$ of $BtoA2$ implements variable $BtoA$ of AB.

We then saw that this means that this theorem is true, where this is the formula we called $SpecH$.

The $AB2$ protocol implements the AB protocol, where $RemoveBad(AtoB2)$ implements $AtoB$ and $RemoveBad(BtoA2)$ implements $BtoA$.

means

$$\text{THEOREM} \; \begin{pmatrix} \land \; Spec \text{ of } AB2 \\ \land \; \Box \land \; AtoB = RemoveBad(AtoB2) \\ \quad\;\; \land \; BtoA = RemoveBad(BtoA2) \end{pmatrix} \Rightarrow \; Spec \text{ of } AB$$

We saw that the $AB2$ protocol implements the AB protocol, where $RemoveBad$ of $AtoB2$ implements variable $AtoB$ of AB, and $RemoveBad$ of $BtoA2$ implements variable $BtoA$ of AB.

We then saw that this means that this theorem is true, where this is the formula we called $SpecH$.

[slide 170]

The $AB2$ protocol implements the AB protocol, where $RemoveBad(AtoB2)$ implements $AtoB$ and $RemoveBad(BtoA2)$ implements $BtoA$.

means

$$\text{THEOREM} \begin{pmatrix} \wedge\ Spec \text{ of } AB2 \\ \wedge\ \Box \wedge AtoB = RemoveBad(AtoB2) \\ \quad\ \ \wedge BtoA = RemoveBad(BtoA2) \end{pmatrix} \Rightarrow Spec \text{ of } AB$$

which in $AB2$ is equivalent to

And we then saw that in module $AB2$ we can write an equivalent assertion as

The $AB2$ protocol implements the AB protocol, where $RemoveBad(AtoB2)$ implements $AtoB$ and $RemoveBad(BtoA2)$ implements $BtoA$.

means

$$\text{THEOREM} \begin{pmatrix} \wedge\ Spec \text{ of } AB2 \\ \wedge\ \Box \wedge\ AtoB = RemoveBad(AtoB2) \\ \quad\ \wedge\ BtoA = RemoveBad(BtoA2) \end{pmatrix} \Rightarrow Spec \text{ of } AB$$

which in $AB2$ is equivalent to

$$\begin{aligned} AB \triangleq \text{INSTANCE } AB \text{ WITH } & AtoB \leftarrow RemoveBad(AtoB2), \\ & BtoA \leftarrow RemoveBad(BtoA2) \end{aligned}$$

$$\text{THEOREM } Spec \Rightarrow AB!Spec$$

And we then saw that in module $AB2$ we can write an equivalent assertion as this INSTANCE statement and theorem.

The $AB2$ protocol implements the AB protocol, where $RemoveBad(AtoB2)$ implements $AtoB$ and $RemoveBad(BtoA2)$ implements $BtoA$.

refinement mapping

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow RemoveBad(AtoB2)$, $BtoA \leftarrow RemoveBad(BtoA2)$

THEOREM $Spec \Rightarrow AB!Spec$

And we then saw that in module $AB2$ we can write an equivalent assertion as this INSTANCE statement and theorem.

These substitutions are called a refinement mapping.

The $AB2$ protocol implements the AB protocol

under the refinement mapping

$$AtoB \leftarrow RemoveBad(AtoB2),$$
$$BtoA \leftarrow RemoveBad(BtoA2)$$

refinement mapping

$AB \triangleq$ INSTANCE AB WITH $AtoB \leftarrow RemoveBad(AtoB2),$ $BtoA \leftarrow RemoveBad(BtoA2)$

THEOREM $Spec \Rightarrow AB!Spec$

And we then saw that in module $AB2$ we can write an equivalent assertion as this INSTANCE statement and theorem.

These substitutions are called a refinement mapping.

And we say that the $AB2$ protocol implements the AB protocol under this refinement mapping.

The $AB2$ protocol implements the AB protocol under the refinement mapping

$$AtoB \leftarrow RemoveBad(AtoB2),$$
$$BtoA \leftarrow RemoveBad(BtoA2)$$

means

$$AB \triangleq \text{INSTANCE } AB \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2),$$
$$BtoA \leftarrow RemoveBad(BtoA2)$$
$$\text{THEOREM } Spec \Rightarrow AB!Spec$$

So that means

The $AB2$ protocol implements the AB protocol under the refinement mapping

$$AtoB \leftarrow RemoveBad(AtoB2),$$
$$BtoA \leftarrow RemoveBad(BtoA2)$$

means

$$AB \triangleq \text{INSTANCE } AB \text{ WITH } AtoB \leftarrow RemoveBad(AtoB2),$$
$$BtoA \leftarrow RemoveBad(BtoA2)$$

$$\text{THEOREM } Spec \Rightarrow AB!Spec$$

So that means that this theorem is true.

And TLC can check the theorem by using a model with $Spec$ as the behavior specification and AB bang $Spec$ as the temporal property to be checked.

If $Spec2$ does not contain all the variables of $Spec1$,

In general, if a specification $Spec2$ does not contain all the variables of a specification $Spec1$, If $Spec2$ does not contain all the variables of $Spec1$, then $Spec2$ can implement $Spec1$ only under a refinement mapping

In general, if a specification $Spec2$ does not contain all the variables of a specification $Spec1$, then $Spec2$ can implement $Spec1$ only under a refinement mapping

If $Spec2$ does not contain all the variables of $Spec1$, then $Spec2$ can implement $Spec1$ only under a refinement mapping that assigns expressions of $Spec2$ to all the variables in $Spec1$ that are not also in $Spec2$.

In general, if a specification $Spec2$ does not contain all the variables of a specification $Spec1$, then $Spec2$ can implement $Spec1$ only under a refinement mapping that assigns expressions of $Spec2$ to all the variables in $Spec1$ that are not also in $Spec2$.

If $Spec2$ does not contain all the variables of $Spec1$, then $Spec2$ can implement $Spec1$ only under a refinement mapping that assigns expressions of $Spec2$ to all the variables in $Spec1$ that are not also in $Spec2$.

This is the usual case.

In general, if a specification $Spec2$ does not contain all the variables of a specification $Spec1$, then $Spec2$ can implement $Spec1$ only under a refinement mapping that assigns expressions of $Spec2$ to all the variables in $Spec1$ that are not also in $Spec2$.

This is the usual case.

If $Spec2$ does not contain all the variables of $Spec1$, then $Spec2$ can implement $Spec1$ only under a refinement mapping that assigns expressions of $Spec2$ to all the variables in $Spec1$ that are not also in $Spec2$.

This is the usual case.

Even if $Spec1$ and $Spec2$ have a variable v in common,

Even if $Spec1$ and $Spec2$ have a variable v in common,

If $Spec2$ does not contain all the variables of $Spec1$,
then $Spec2$ can implement $Spec1$ only under a
refinement mapping that assigns expressions of
$Spec2$ to all the variables in $Spec1$ that are not
also in $Spec2$.

This is the usual case.

Even if $Spec1$ and $Spec2$ have a variable v in common,
the refinement mapping might substitute an expression
of $Spec2$ other than v for the variable v of $Spec1$.

Even if $Spec1$ and $Spec2$ have a variable v in common,
the refinement mapping might substitute an expression of $Spec2$ other than v for the variable v of $Spec1$.

What does it mean for a program to implement a TLA$^+$ specification $Spec$?

What does it mean for a program written in a programming language to implement a TLA$^+$ specification $Spec$?

What does it mean for a program to implement a TLA+ specification $Spec$?

It means that we can, in principle, write a TLA+ specification $SpecPgm$ of the program,

It means that we can, in principle, write a TLA+ specification $SpecPgm$ of the program, What does it mean for a program to implement a TLA$^+$ specification $Spec$?

It means that we can, in principle, write a TLA$^+$ specification $SpecPgm$ of the program, and $SpecPgm$ implements $Spec$ under a suitable refinement mapping.

It means that we can, in principle, write a TLA$^+$ specification $SpecPgm$ of the program, and $SpecPgm$ implements $Spec$ under a suitable refinement mapping.

What does it mean for a program to implement a TLA$^+$ specification $Spec$?

It means that we can, in principle, write a TLA$^+$ specification $SpecPgm$ of the program, and $SpecPgm$ implements $Spec$ under a suitable refinement mapping.

We can't do that in practice,

It means that we can, in principle, write a TLA$^+$ specification $SpecPgm$ of the program, and $SpecPgm$ implements $Spec$ under a suitable refinement mapping.

We can't do that in practice because $SpecPgm$ would be much too long and complicated, What does it mean for a program to implement a TLA+ specification $Spec$?

It means that we can, in principle, write a TLA+ specification $SpecPgm$ of the program, and $SpecPgm$ implements $Spec$ under a suitable refinement mapping.

We can't do that in practice, but understanding refinement mappings can help prevent implementation errors.

It means that we can, in principle, write a TLA+ specification $SpecPgm$ of the program, and $SpecPgm$ implements $Spec$ under a suitable refinement mapping.

We can't do that in practice because $SpecPgm$ would be much too long and complicated, but understanding refinement mappings can help prevent implementation errors.

Even if you can't write the refinement mapping in TLA$^+$,

Even if you can't write the refinement mapping in TLA$^+$,

Even if you can’t write the refinement mapping in TLA$^+$, you should be able to explain informally how the spec’s variables are implemented by the program’s state.

Even if you can’t write the refinement mapping in TLA$^+$, you should be able to explain informally how the spec’s variables are implemented by the program’s state.

Even if you can't write the refinement mapping in TLA$^+$, you should be able to explain informally how the spec's variables are implemented by the program's state.

The informal refinement mapping explains what the program is doing.

Even if you can't write the refinement mapping in TLA$^+$, you should be able to explain informally how the spec's variables are implemented by the program's state.

The informal refinement mapping explains what the program is doing.

Even if you can't write the refinement mapping in TLA$^+$, you should be able to explain informally how the spec's variables are implemented by the program's state.

The informal refinement mapping explains what the program is doing.

Writing it down can expose errors in the program.

Even if you can't write the refinement mapping in TLA$^+$, you should be able to explain informally how the spec's variables are implemented by the program's state.

The informal refinement mapping explains what the program is doing.

Writing it down, perhaps as comments in the code, can expose errors in the program.

IMAGINARY VARIABLES

We added imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol spec to obtain $SpecP$.

We added the imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol specification to obtain specification $SpecP$.

We added imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol spec to obtain $SpecP$.

We did that to write a desired liveness property.

We added the imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol specification to obtain specification $SpecP$.

We did that in order to write a desired liveness property.

We added imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol spec to obtain $SpecP$.

We added imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec to obtain $SpecH$.

We added the imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol specification to obtain specification $SpecP$.

We did that in order to write a desired liveness property.

We added imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol specification to obtain specification $SpecH$.

We added imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol spec to obtain $SpecP$.

We added imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec to obtain $SpecH$.

We did that to show $AB2$ implements AB.

We did that in order to show that the $AB2$ protocol's safety spec implements the AB protocol's safety spec.

We added imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol spec to obtain $SpecP$.

We added imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec to obtain $SpecH$.

$Spec2$ obtained by adding imaginary variables to $Spec1$

We did that in order to show that the $AB2$ protocol's safety spec implements the AB protocol's safety spec.

In general, a specification $Spec2$ is obtained by adding imaginary variables to a specification $Spec1$

We added imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol spec to obtain $SpecP$.

We added imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec to obtain $SpecH$.

$Spec2$ obtained by adding imaginary variables to $Spec1$ means $Spec2$ and $Spec1$ allow the same behaviors

We did that in order to show that the $AB2$ protocol's safety spec implements the AB protocol's safety spec.

In general, a specification $Spec2$ is obtained by adding imaginary variables to a specification $Spec1$

means that $Spec2$ and $Spec1$ allow the same behaviors

We added imaginary variables $AtoBgood$ and $BtoAgood$ to the $AB2$ protocol spec to obtain $SpecP$.

We added imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec to obtain $SpecH$.

$Spec2$ obtained by adding imaginary variables to $Spec1$ means $Spec2$ and $Spec1$ allow the same behaviors if we ignore the values of the imaginary variables.

We did that in order to show that the $AB2$ protocol's safety spec implements the AB protocol's safety spec.

In general, a specification $Spec2$ is obtained by adding imaginary variables to a specification $Spec1$

means that $Spec2$ and $Spec1$ allow the same behaviors

if we ignore the values of the imaginary variables.

We added imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec to obtain $SpecH$.

We did that to show $AB2$ implements AB.

We added imaginary variables to show that the $AB2$ spec implements the AB spec.

We added imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec to obtain $SpecH$.

We did that to show $AB2$ implements AB.

This wasn't necessary because we could use a refinement mapping instead.

We added imaginary variables to show that the $AB2$ spec implements the AB spec.

This wasn't necessary because we were able to use a refinement mapping instead.

We added imaginary variables $AtoB$ and $BtoA$
to the $AB2$ protocol spec to obtain $SpecH$.

We did that to show $AB2$ implements AB .

This wasn't necessary because we could use a
refinement mapping instead.

Sometimes we have to add imaginary variables to define a
refinement mapping.

We added imaginary variables to show that the $AB2$ spec implements the AB spec.

This wasn't necessary because we were able to use a refinement mapping instead.

But sometimes we have to add imaginary variables in order to define a refinement mapping.

The AB and $AB2$ protocols are essentially the same (ignoring liveness).

The AB and $AB2$ protocols are essentially the same, if we ignore liveness.

The AB and $AB2$ protocols are essentially the same (ignoring liveness).

So $Spec$ of AB should implement $Spec$ of $AB2$ under a refinement mapping.

The AB and $AB2$ protocols are essentially the same, if we ignore liveness.

So specification $Spec$ of module AB should implement specification $Spec$ of module $AB2$ under a refinement mapping.

The AB and $AB2$ protocols are essentially the same (ignoring liveness).

So $Spec$ of AB should implement $Spec$ of $AB2$ under a refinement mapping.

Showing this requires adding to module AB

Showing this requires adding to module AB :

The AB and $AB2$ protocols are essentially the same (ignoring liveness).

So $Spec$ of AB should implement $Spec$ of $AB2$ under a refinement mapping.

Showing this requires adding to module AB

$$AB2 \triangleq \textsc{instance}\ AB2\ \textsc{with}\ AtoB2 \leftarrow \ldots,\ BtoA2 \leftarrow \ldots$$

Showing this requires adding to module AB :

An INSTANCE statement giving the refinement mapping

The AB and $AB2$ protocols are essentially the same (ignoring liveness).

So $Spec$ of AB should implement $Spec$ of $AB2$ under a refinement mapping.

Showing this requires adding to module AB

$AB2 \triangleq$ INSTANCE $AB2$ WITH $AtoB2 \leftarrow \ldots,\ BtoA2 \leftarrow \ldots$

THEOREM $Spec \Rightarrow AB2!Spec$

Showing this requires adding to module AB :

An INSTANCE statement giving the refinement mapping and checking this theorem.

The AB and $AB2$ protocols are essentially the same (ignoring liveness).

So $Spec$ of AB should implement $Spec$ of $AB2$ under a refinement mapping.

Showing this requires adding to module AB

$$AB2 \triangleq \text{INSTANCE } AB2 \text{ WITH } AtoB2 \leftarrow \boxed{\ldots},\ BtoA2 \leftarrow \boxed{\ldots}$$

$$\text{THEOREM } Spec \Rightarrow AB2!Spec$$

expressions of module AB

These expressions of the refinement mapping must be written in terms of the variables of module AB.

The AB and $AB2$ protocols are essentially the same (ignoring liveness).

So $Spec$ of AB should implement $Spec$ of $AB2$ under a refinement mapping.

Showing this requires adding to module AB

$AB2 \triangleq$ INSTANCE $AB2$ WITH $AtoB2 \leftarrow \boxed{\ldots}$, $BtoA2 \leftarrow \boxed{\ldots}$

THEOREM $Spec \Rightarrow AB2!Spec$

expressions of module AB

Impossible without adding imaginary variables to $Spec$ of AB that remember where messages were lost from $AtoB$ and $BtoA$.

These expressions of the refinement mapping must be written in terms of the variables of module AB.

This is impossible without adding imaginary variables to specification $Spec$ of module AB that remember where messages that were lost from the message sequences $AtoB$ and $BtoA$ used to be.

Imaginary Variables

Imaginary Variables

Imaginary Variables

– Need not describe actual state of the system.

Imaginary Variables

Need not describe any actual state of the system.

Imaginary Variables

– Need not describe actual state of the system.

If their values can be described in terms of the original variables, then they are unnecessary.

Imaginary Variables

Need not describe any actual state of the system.

In fact, if their values can be described in terms of the original variables that describe the actual state, then the imaginary variables are unnecessary.

Imaginary Variables

– Need not describe actual state of the system.

If their values can be described in terms of the original variables, then they are unnecessary.

We didn't need to add imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol to show it implements the AB protocol

For example, we didn't need to add imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec in order to show that it implements the AB protocol spec

Imaginary Variables

– Need not describe actual state of the system.

If their values can be described in terms of the original variables, then they are unnecessary.

We didn't need to add imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol to show it implements the AB protocol because we could specify their values with a refinement mapping.

For example, we didn't need to add imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec in order to show that it implements the AB protocol spec because we could specify the values of those variables of the AB spec with a refinement mapping.

Imaginary Variables

– Need not describe actual state of the system.

– Are not meant to be implemented.

For example, we didn't need to add imaginary variables $AtoB$ and $BtoA$ to the $AB2$ protocol spec in order to show that it implements the AB protocol spec because we could specify the values of those variables of the AB spec with a refinement mapping.

Imaginary variables are not meant to be implemented.

Imaginary Variables

– Need not describe actual state of the system.

– Are not meant to be implemented.

– May be needed to construct a refinement mapping.

And imaginary variables may be needed to construct a refinement mapping.

Imaginary Variables

– Need not describe actual state of the system.

– Are not meant to be implemented.

– May be needed to construct a refinement mapping.

And imaginary variables may be needed to construct a refinement mapping.

Imaginary variables

Auxiliary ~~Imaginary~~ Variables

– Need not describe actual state of the system.

– Are not meant to be implemented.

– May be needed to construct a refinement mapping.

And imaginary variables may be needed to construct a refinement mapping.

Imaginary variables are usually called *auxiliary* variables.

Auxiliary

~~Imaginary~~ Variables

– Need not describe actual state of the system.

– Are not meant to be implemented.

– May be needed to construct a refinement mapping.

You can learn more about them by stopping the video and downloading the paper

Auxiliary Variables in TLA+

And imaginary variables may be needed to construct a refinement mapping.

Imaginary variables are usually called *auxiliary* variables.

You can learn more about them by stopping the video and downloading this paper

WHAT'S NEXT ?

This is the last lecture of the TLA^+ Video Course.

This is the last lecture of the TLA^+ Video Course.

This is the last lecture of the TLA$^+$ Video Course.

You’re now ready to write your own specs, including liveness conditions, and to show that one spec implements another.

This is the last lecture of the TLA$^+$ Video Course.

You’re now ready to write your own specs, including liveness conditions, and to show that one spec implements another.

This is the last lecture of the TLA+ Video Course.

You're now ready to write your own specs, including liveness conditions, and to show that one spec implements another.

It may not be easy at first.

This is the last lecture of the TLA+ Video Course.

You're now ready to write your own specs, including liveness conditions, and to show that one spec implements another.

It may not be easy at first.

This is the last lecture of the TLA$^+$ Video Course.

You're now ready to write your own specs, including liveness conditions, and to show that one spec implements another.

It may not be easy at first.

Writing good specs takes practice.

Writing good specs takes practice.

This is the last lecture of the TLA+ Video Course.

You're now ready to write your own specs, including liveness conditions, and to show that one spec implements another.

It may not be easy at first.

Writing good specs takes practice.
Reading other people's specs can help.

Writing good specs takes practice.
Reading other people's specs can help.

This is the last lecture of the TLA+ Video Course.

You're now ready to write your own specs, including liveness conditions, and to show that one spec implements another.

It may not be easy at first.

Writing good specs takes practice.
Reading other people's specs can help.

I hope the TLA+ web pages will eventually contain many examples of realistic specs.

Writing good specs takes practice.
Reading other people's specs can help.

I hope the TLA+ web pages will eventually contain many examples of realistic specs.

There is still plenty to learn about TLA$^+$ and its tools:

There is still plenty for you to learn about TLA$^+$ and its tools:

There is still plenty to learn about TLA$^+$ and its tools:

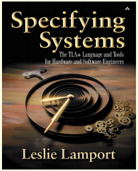
– A few TLA$^+$ features.

There is still plenty for you to learn about TLA$^+$ and its tools:

There are a few TLA$^+$ features that you haven’t seen.

There is still plenty to learn about TLA+ and its tools:

– A few TLA+ features.

You can find out about them by browsing: 

There is still plenty for you to learn about TLA+ and its tools:

There are a few TLA+ features that you haven't seen.

You can find out about them by browsing the *Specifying Systems* book.

There is still plenty to learn about TLA+ and its tools:

– A few TLA+ features.

– Many Toolbox features.

There is still plenty for you to learn about TLA+ and its tools:

There are a few TLA+ features that you haven't seen.

You can find out about them by browsing the *Specifying Systems* book.

There are many Toolbox features that I haven't shown you.

There is still plenty to learn about TLA+ and its tools:

– A few TLA+ features.

– Many Toolbox features.

You can find out about them by browsing
the Toolbox's help pages.

There is still plenty for you to learn about TLA+ and its tools:

There are a few TLA+ features that you haven't seen.

You can find out about them by browsing the *Specifying Systems* book.

There are many Toolbox features that I haven't shown you.

You can find them by browsing the Toolbox's help pages.

There is still plenty to learn about TLA$^+$ and its tools:

- A few TLA$^+$ features.
- Many Toolbox features.
- PlusCal

And there's the PlusCal algorithm language,

There is still plenty to learn about TLA$^+$ and its tools:

- A few TLA$^+$ features.
- Many Toolbox features.
- PlusCal — A language for writing TLA$^+$ specs that look more familiar to programmers.

And there's the PlusCal algorithm language,

A language for writing TLA+ specs that look more familiar to programmers.

There is still plenty to learn about TLA$^+$ and its tools:

– A few TLA$^+$ features.

– Many Toolbox features.

– PlusCal A language for writing TLA$^+$ specs that look more familiar to programmers.

See the TLA$^+$ Web site for documentation.

And there's the PlusCal algorithm language,

A language for writing TLA+ specs that look more familiar to programmers.

See the TLA$^+$ Web site for documentation.

This is the end of the course. You've come a long way – perhaps further than you realize. As you go forward, remember to take the time to stop and think. I hope what you've learned here will help you do that.

End of Lecture 10, Part 2

IMPLEMENTATION WITH REFINEMENT

REFINEMENT MAPPINGS